2

MASSACHUSETTS INSTITUTE OF TECHNOLOGY VLSI PUBLICATIONS

VLSI Memo No. 89-546
May 1989

Fast Computation Using Faulty Hypercubes

Johan Hastad, Tom Leighton, and Mark Newman

Abstract

We consider the computational power of a hypercube containing a potentially large number of randomly located faulty components. We describe a randomized algorithm which embeds an N-node hypercube in an N-node hypercube with faulty processors. Provided that the processors of the N-node hypercube are faulty with probability $p < 1$, and that the faults are independently distributed, we show that with high probability, the faulty hypercube can emulate the fault-free hypercube with only constant slowdown. In other words, an N-node hypercube with faults can simulate T steps of an N-node fault-free hypercube in $O(T)$ steps. The embedding is easy to construct in polylogarithmic time using only local control. We also describe $O(\log N)$-step routing algorithms which ensure the delivery of messages with high probability even when a constant fraction of the nodes and edges have failed. The routing results represent the first adaptive routing algorithms for which an effective theoretical analysis has been achieved.

Acknowledgements

To appear in the *ACM Symposium on Theory of Computing*, May 1989. This research was supported in part by the Defense Advanced Research Projects Agency under contract numbers N00014-80-C-0622 and N00014-87-K-0825, the Air Force under contract number AFOSR-86-0078, and the Army under contract number DAAL03-86-K-0171. Leighton was supported in part by a NSF Presidential Young Investigator Award with matching funds from IBM Corporation.

Author Information

Hastad, *current address:* Royal Institute of Technology, Stockholm, Sweden.
Leighton: Department of Mathematics and Laboratory for Computer Science, Room NE43-332, MIT, Cambridge, MA 02139. (617) 253-5876.
Newman: Department of Mathematics and Laboratory for Computer Science, Room NE43-334, MIT, Cambridge, MA 02139. (617) 253-5866

Fast Computation Using Faulty Hypercubes

(extended abstract)

Johan Hastad [*] Tom Leighton [†] Mark Newman [‡]

Abstract

We consider the computational power of a hypercube containing a potentially large number of randomly located faulty components. We describe a randomized algorithm which embeds an N-node hypercube in an N-node hypercube with faulty processors. Provided that the processors of the N-node hypercube are faulty with probability $p < 1$, and that the faults are independently distributed, we show that with high probability, the faulty hypercube can emulate the fault-free hypercube with only constant slowdown. In other words, an N-node hypercube with faults can simulate T steps of an N-node fault-free hypercube in $O(T)$ steps. The embedding is easy to construct in polylogarithmic time using only local control. We also describe $O(\log N)$-step routing algorithms which ensure the delivery of messages with high probability even when a constant fraction of the nodes and edges have failed. The routing results represent the first adaptive routing algorithms for which an effective theoretical analysis has been achieved.

1 Introduction

The hypercube is emerging as one of the most effective and popular network architectures for large scale parallel computers. Already, hypercube-based machines containing 2^{16} processing elements have been manufactured and sold commercially, and it is possible that in the not-too-distant future, hypercube-based machines containing millions of processors will be available. A main concern with the development of such large scale systems is fault tolerance. In particular, it is crucial that such a system be able to withstand an accumulation of faults among a reasonable number of its components.

In this paper, we investigate the tolerance of the hypercube to randomly distributed faults Our results are very encouraging and, in some areas, striking. For example, provided that the processors of the N-node hypercube are faulty with probability $p < 1$, and that the faults are independently distributed, we show that, with high probability, the faulty hypercube can simulate a fault-free N-node hypercube with only constant slowdown. Of course, the constant factor slowdown depends on p (since it must always be at least $\frac{1}{1-p}$), but it does not depend on N. Moreover, the algorithm for reconfiguring the faulty hypercube is simple, needs only local control and runs in only a polylogarithmic number of steps. Hence the faulty hypercube can quickly reconfigure itself to appear fault-free (except for the constant slowdown) without the intervention of a central control.

The reconfiguration algorithm for faulty hypercubes described in this paper represents a significant improvement over previous work. In particular, although some known reconfiguration algorithms preserve locality (i.e., neighbors in the virtual fault-free hypercube are simulated by nodes within constant distance of one another in the faulty hypercube), the best previous simulation algorithms used off-line and nonconstructive reconfiguration strategies to obtain slowdowns of size $\Theta(\sqrt{\log\log N})$ (see [HLN]). The best previous on-line algorithms result in slowdowns of size $\Theta(\sqrt{\log N})$ (also in [HLN]). The main problem with these algorithms is that they require a nonconstant number of wires in the virtual fault-free hypercube to be routed across a single wire of the faulty hypercube. Hence, a constant time simulation using these algorithms is not possible. Otherwise, there has been relatively little previous work on reconfiguring faulty hypercubes that contain more than a few faults. The only exceptions (of which we are aware) are the work of Becker and Simon ([BS]) and Graham, Harary, Livingston and Stout ([GHLS]), who consider fault-free subcubes of a hypercube containing worst case faults. The constraint that the embedded cube be a subcube (i.e. dilation 1) is very restrictive, as is the assumption that faults are located in a worst case fashion. Hence, the techniques and results of [BS] and [GHLS] are quite different from those presented here. Dolev, Halpern, Simons and Strong ([DHSS]) also con-

[*]Royal Institute of Technology, Stockholm SWEDEN.

[†]Math Dept. and Lab for Comp. Sci., MIT, Cambridge, MA. Supported by Air Force contract AFSOR-86-0078, DARPA contract N00014-80-C-0622, Army contract DAAL03-86-K-0171, and an NSF Presidential Young Investigator Award with matching funds from IBM.

[‡]Math Dept. and Lab for Comp. Sci., MIT, Cambridge, MA. Supported by DARPA contract N00014-80-C-0622.

sidered faults in a worst case model. Routes were chosen from a predetermined set of paths. Their techniques and results also differ markedly from ours.

In the latter sections of the paper, we consider the problem of packet routing on a hypercube with faults. Of course, once we have reconfigured a faulty hypercube, we can simply use the classical algorithms to route any permutation of N packets in $O(\log N)$ steps. However, the question of routing in a fault tolerant fashion directly on a faulty hypercube is still of some interest. For example, can we handle faults that are occurring dynamically without having to reconfigure all the time? Also, are there efficient routing algorithms that are adaptive in the sense that packets can be routed locally around faults without knowing in advance where the faulty components lie? Affirmative answers to these questions could well be of interest in practice and in theory since all known $O(\log N)$ step routing algorithms on a hypercube are inherently non-adaptive.

Motivated by such issues, we describe and analyze a randomized packet routing algorithm that adaptively routes packets around faults as they are encountered in an N-node hypercube that contains $\Theta(N)$ randomly located faulty nodes and $\Theta(N \log N)$ randomly located faulty edges. We prove that the algorithm routes any permutation on the working processors in $O(\log N)$ steps with high probability. Packets that start or end at faulty nodes are eventually determined to be undeliverable. All the deliverable packets arrive at their destinations provided that they are not located in the immediate vicinity of a processor at the moment when it fails. The algorithm is fault-tolerant in the sense that no advance knowledge of the locations of the faults is needed for the path selection, but it is not completely tolerant of nodes which fail while holding packets. The algorithm is of interest because during most steps, few processors will fail and almost all deliverable packets will be delivered. In addition, the algorithm itself is quite simple and is the first adaptive routing algorithm for which an $O(\log N)$ bound on the routing time has been achieved.

There has been some previous work on packet routing in faulty hypercubes. Most notably, Rabin ([R]) has devised an elegant scheme wherein each packet to be routed is decomposed into $\Theta(\log N)$ pieces. The pieces are routed in a randomized nonadaptive fashion to their destinations and then recombined to form the original message. A key aspect of the scheme is that the packet decomposition uses error-correcting codes. Therefore only a constant fraction of the pieces of any packet need to get through to the destination for the packet to be reconstructed. Such a scheme can be useful if the original packets represent relatively long bit streams In particular, the original packets must have length $\Omega(\log^2 N)$. Additionally, at most $O(\frac{N}{\log N})$ edge faults can be absorbed. Under these conditions, the Rabin algorithm provides a fully fault-tolerant routing of N packets in $O(\log N)$ steps with high probability.

In the paper, we show how the Rabin result can b achieved with a simpler algorithm and analysis. Ou analysis permits the routing to absorb up to $O(N)$ edge faults as well as $O(\frac{N}{\log N})$ node faults. (A similar result with a more complicated algorithm and analysis has recently been discovered by Giladi ([G]).) We also briefly sketch a way to potentially improve its tolerance to faults in as many as a constant fraction of components by combining the decomposition scheme with our adaptive algorithm for routing around faults.

Before describing our results further, it is useful to review some terminology. First, we describe the n-dimensional, N-node hypercube H_n, $N = 2^n$. The nodes of H_n are denoted by n-bit binary strings, and two nodes are linked by an edge if the associated strings differ in precisely one bit. If the differing bit is in the i^{th} position $(1 \le i \le n)$ then the associated edge is called a *dimension i edge*. The neighbor of a node v across the i^{th} dimension will be denoted by v^i. Similarly $v^{i_1 i_2 \ldots i_k}$ will denote the node reached from v by traversing dimensions $i_1, i_2, \ldots, i_k$. For the rest of this paper we will use n and $\log N$ interchangeably.

An *embedding* of a virtual fault-free hypercube H'_n into a faulty hypercube H_n is a map $\phi : H'_n \rightarrow H_n$ that maps nodes of H'_n to functioning nodes of H_n and edges of H'_n to functioning paths of H_n. In this paper, a path is said to be functioning if all the nodes and edges on th path are fault-free. (In some other papers ([HLN], [A]), there are models which allow the nodes on the path to be faulty.) The *dilation* of an embedding is the length of the longest path $\phi(e)$ in H_n corresponding to an edge in H'_n. The *load* of an embedding is the maximum number of nodes of H'_n mapped to a single node of H_n. The *congestion* of an embedding is the maximum number of paths $\phi(e)$ that use a single edge of H_n. Expressed in these terms, our task is to find an embedding of H'_n in H_n with constant dilation, load and congestion. Once this is done, then H_n will be able to simulate H'_n with constant slowdown.

In this paper, we will consider a fault model where nodes and edges [illegible], fail independently with probability $p < 1$. For most of the proofs, we will focus on the case where $p < .1$ and only nodes fail. We later explain how to handle larger probabilities of failure and edge failures. The case where $p < .1$ is easier because each node will then have a reasonably large neighborhood of functioning processors with high probability.

In reality, it is unlikely that so many processors will fail in a short period of time. However, our results are all upper bounds, and the case where only a small number of components fail is even easier. Also, if a hypercube-based machine is inaccessible or vulnerable to catastrophe then it may be possible that a large number of faults occur over time. This approach can also be applied in a hierarchical fashion if the faults are not independently located. For example, if entire subcubes are likely to fail as a unit (e.g. an entire chip, board or box fails) but units fail independently, then the same results apply. Even if a few locally concentrated faults occur, our results apply to the affected region. This analysis cannot apply to a worst case allocation of faults, since by selectively killing $\Theta(N)$ nodes we can disconnect the hypercube into components of size $O(\frac{N}{\sqrt{\log N}})$.

Our proofs use a fair amount of probabilistic and combinatorial analysis. Although no individual step in the analysis is particularly difficult or noteworthy, there are some approaches to these problems that may be of use with other hypercube-related problems. In particular, there is one simple observation that is used in various forms throughout the paper. Although the observation has probably been made by others, it is basic enough that we think it worth highlighting as a paradigm for distributed match-making.

We will describe the result in its most basic form. Consider a collection of $\Theta(N)$ men and $\Theta(N)$ women at a dance. Assume that each man has at least $\Omega(X)$ female friends and that each woman has at most $O(X)$ male friends. By Hall's marriage theorem, it is possible to schedule $O(1)$ rounds of dances so that every man dances with at least one friend and every woman dances at most $O(1)$ times. Unfortunately, the problem of scheduling dance partners requires substantial global coordination. For our purposes, we focus on a scenario where pairing is accomplished simply by a man asking a woman to dance. If many men ask a woman to dance at once, she accepts as many as she can, making sure not to exceed her capacity of $C = O(1)$ dances for the evening. If she can only accept some of the men, she prefers the tallest among them. Each man chooses a friend randomly for each dance (without knowledge of which women are tired or which women other men are asking) until he dances. The result (which we call the Dance Hall Theorem - pun intended) is that if $X = \Theta(\log N)$, and there are $\Omega(\log N)$ dances, then with high probability (i.e. with probability exceeding $1 - O(\frac{1}{N^c})$ for some constant $c > 1$) every man will dance during the course of the evening.

The Dance Hall Theorem scenario first arises in our analysis when we attempt to embed the nodes of H'_n in the functioning nodes of H_n. The nodes of H'_n correspond to men and the functioning nodes of H_n correspond to women. If a man dances with a woman, then the corresponding node of H'_n will be simulated by the corresponding node of H_n. We need the Dance Hall Theorem to ensure that the load of the embedding is $O(1)$ (i.e. every woman dances with $O(1)$ men) and to ensure that the embedding can be constructed quickly with local control (no global matchmaker). We also need some other as-yet-undescribed properties of the Dance Hall Theorem schedule to ensure that the dilation and congestion of the embedding are $O(1)$, but these are more technical in nature and will be dealt with in the main text.

The technical portion of the paper is divided into three sections. In section 2, we present the constant delay embedding. Section 3 contains the $O(\log N)$ time adaptive routing algorithm and in section 4 we show how to improve Rabin's fault-tolerant results with a simpler algorithm.

2 Constant Delay Reconfiguration

To achieve a constant delay embedding, we need the load, dilation and congestion to all be constant. The embedding we will find will have a load and congestion which depend strongly on the probability of failure - clearly the more nodes that fail, the more nodes that have to be simulated by any one processor. However, the dilation will always remain five, and each processor will be simulated by one of its neighbors, provided that $p < 1 - \sqrt[4]{.5}$ (about .16).

In order to simplify the analysis, each node (live or dead) finds a neighbor to simulate it. We first assign nodes to live neighbors so that no node simulates more than a constant number of its neighbors. Then each pair of nodes simulating neighbors finds a live path between them of length five so that no more than a constant number of these paths congest any edge. We will use two similar algorithms to accomplish these two tasks.

Let A_p and s_p be constants (to be determined later) which depend only upon the probability p of failure. Call a node ***unsaturated*** if it is live and if it has been assigned to simulate fewer than A_p of its neighbors. Otherwise, it is ***saturated***.

Throughout, we will assume some fixed ordered labeling of the nodes. The most convenient one is the lexicographical order of the labels. The assignment algorithm proceeds in rounds. During a round, a previously unsaturated node might be picked by enough unassigned nodes so as to exceed its capacity A_p. In such a case, we require the node to accept enough of the simulation requests to saturate it. All accepted nodes should have lower labels than those which are rejected.

Algorithm 2.1 performs the first phase:

```
for $i = 1$ to $s_p n$
    for each unassigned node $w$
        $w$ picks one of its neighbors uniformly
    each unsaturated node $v$ agrees to simulate as
    many nodes as it can without exceeding its capacity, giving preference to lower labeled nodes
    all excess nodes remain unassigned
```

Figure 1: Algorithm 2.1

Since the algorithm never assigns a saturated node to simulate another node, no node simulates more than A_p nodes. Thus, a constant load embedding results.

To facilitate our proofs, we will first formulate a sequential algorithm similar to Algorithm 2.1. We will prove that this new algorithm assigns to each node a neighboring node to simulate it. We will then show that, except for a small proportion of executions, the algorithms behave the same.

In each round of Algorithm 2.2, unassigned nodes act sequentially. Each node chooses a neighbor to simulate it only after all lower labeled nodes have chosen. We would like to ensure that all nodes have a large number of choices that will result in a successful assignment. Let α_p depend only upon the probability p. If some node w has fewer than $\alpha_p n$ unsaturated neighbors to choose from during its turn, we designate an arbitrary set of saturated neighbors as *dedicated* to w during its turn. If w chooses a dedicated node during that particular turn, the dedicated node agrees to simulate w even though it is saturated. We dedicate enough nodes so that w has at least $\alpha_p n$ neighbors which, if chosen, will agree to simulate it.

```
for $i = 1$ to $s_p n$
    for unassigned nodes $w$ in lexicographic order
        if $w$ has fewer than $\alpha_p n$ unsaturated neighbors
            arbitrarily dedicate enough (saturated)
            neighbors
        $w$ picks one of its neighbors uniformly
        if the chosen node is unsaturated or dedicated
            $w$ is assigned to that node
        else $w$ remains unassigned
```

Figure 2: Algorithm 2.2

We will show below that with high probability no nodes are ever dedicated during Algorithm 2.2. In that case, the result is the same whether unassigned nodes choose sequentially or in parallel, provided preference goes to the lower labeled nodes. Thus we will show that Algorithms 2.1 and 2.2 produce the same output.

The following lemma proves that Algorithm 2.2 terminates quickly.

Lemma 2.1. *With high probability all nodes have been assigned after $s_p n$ steps of Algorithm 2.2, for sufficiently large s_p.*

Proof. Because each node always has at least $\alpha_p n$ neighbors which will simulate it if chosen, the probability that a given node is assigned during some step is at least α_p, independent of what has occurred in previous steps. Thus the probability that a node remains unassigned after $s_p n$ steps is no more than $(1-\alpha_p)^{s_p n}$. This quantity is less than $\frac{1}{N^k}$ as long as $s_p > \frac{k}{\alpha_p}$. ∎

The following two lemmas show that with high probability Algorithm 2.2 never dedicates saturated nodes. Thus with high probability Algorithms 2.1 and 2.2 behave identically. This proves that Algorithm 2.1 assigns all nodes with high probability. Similar reasoning proves the Dance Hall Theorem described in the introduction.

Lemma 2.2. *For $p < .16$, there exists an ϵ_p such that with high probability each node has at least $\epsilon_p n$ live neighbors.*

Proof. The probability that a node has fewer than ϵn live neighbors equals

$$\sum_{i=0}^{\epsilon n} \binom{n}{i} (1-p)^i p^{n-i}$$

Since the ratio of consecutive terms is always greater than $\frac{1-p}{p}$, this sum is bounded by a constant times its last term. That term is

$$\binom{n}{\epsilon n}(1-p)^{\epsilon n} p^{(1-\epsilon)n} \leq \binom{n}{\epsilon n} p^{(1-\epsilon)n}$$

The second term in the product can be made less than N^{-1-c} for some c by taking ϵ small enough. The first term in the product can be made less than $N^{\frac{c}{2}}$ by taking ϵ small enough as well. The probability that some node has too few neighbors is bounded by the sum of the probabilities for the individual nodes. This multiplies the above bound by N. Thus for any ϵ below both of these thresholds, the theorem applies. ∎

Lemma 2.3. *Given a failure rate p, with high probability a given node v never has fewer than $\alpha_p n$ unsaturated neighbors available during Algorithm 2.2, for $\alpha_p = \frac{\epsilon_p}{2}$.*

Proof. For v to have fewer than $\alpha_p n$ unsaturated neighbors at some point during algorithm 2.2, at least $(\epsilon_p - \alpha_p)n = \alpha_p n$ of v's neighbors must have become saturated during the course of the algorithm.

Each node always has at least $\alpha_p n$ neighbors (including dedicated nodes) to which it might be assigned during any step. Further, if it is assigned, it is equally likely to be assigned to any one of those neighbors. Thus no node has a probability greater than $\frac{1}{\alpha_p n}$ that it will be assigned to any given neighbor, no matter what other assignments have been made previously.

There are no more than n^2 nodes which might be assigned to some node in v's neighborhood. The probability that at least $\alpha_p n$ of v's neighbors become saturated is thus no more than

$$\binom{n^2}{A_p\alpha_p n}\left(\frac{2}{\alpha_p n}\right)^{A_p\alpha_p n} \leq \left(\frac{2e}{A_p\alpha_p^2}\right)^{A_p\alpha_p n}.$$

To saturate $\alpha_p n$ of v's neighbors, there must be at least $A_p\alpha_p n$ nodes at Hamming distance two from v each of which is assigned to a neighbor of v. The first factor in the product represents the number of ways to choose these nodes. Each one of these nodes has at most two neighbors of v to which it might be assigned. Thus the second factor upper-bounds the probability that each of the $A_p \alpha_p n$ nodes is actually assigned to a neighbor of v. Although the probabilities of such selections are technically dependent, the probability a given node is assigned to a neighbor of v is at most $\frac{2}{\alpha_p n}$, no matter what choices the other nodes made.

For A_p large enough, this quantity is an inverse polynomial in N. ■

Lemma 2.3 implies that with high probability Algorithms 2.1 and 2.2 behave identically. We know that Algorithm 2.2 successfully assigns each node to a neighbor with high probability and that Algorithm 2.1 never assigns more than A_p nodes to any node. We conclude that Algorithm 2.1 achieves a constant load embedding with high probability.

Once we've assigned simulating nodes, we need to find paths to simulate the edges in the hypercube. Say that v^b simulates v and $v^{kb'}$ simulates v^k. Then to simulate the edge (v, v^k), the nodes v^b and $v^{kb'}$ choose a path between them of the form $P(v, v^k, b, b', r) = (v^b, v^{br}, v^r, v^{rk}, v^{rkb'}, v^{kb'})$. To avoid ambiguity, we will refer to the choice of r as if it were made by v and v^k even though v^b and $v^{kb'}$ actually choose.

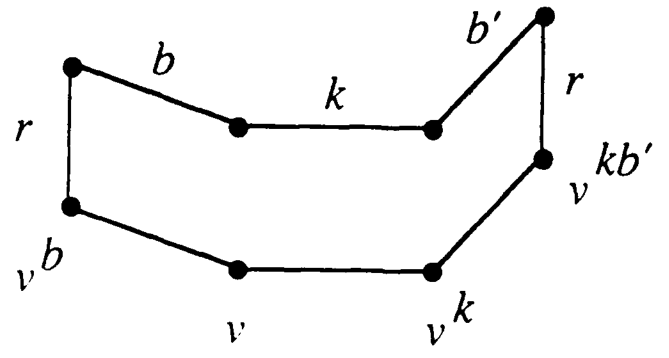

Figure 3: A Choice of Live Path

For two adjacent nodes v and v^k, let $S(v, v^k, b, b')$ be the set of dimensions $r \neq k$ for which $P(v, v^k, b, b', r)$ is a live path. Because $p < 1 - \sqrt[4]{.5}$, there is a chance $(1-p)^4 = s > \frac{1}{2}$ that any given path $P(v, v^k, b, b', r)$ is live. Note that the paths $P(v, v^k, b, b', r)$ $(r \neq k)$ are node-disjoint for a fixed choice of v, v^k, b and b'. Thus the probability that any one of them is live is independent of the other paths.

Lemma 2.4. *With high probability, for all quadruples* (v, v^k, b, b'), $|S(v, v^k, b, b')| > \eta_p n$ *for some constant* η_p.

Proof. Same as lemma 2.2, except that there are $N \log^3 N$ different quadruples. ■

With high probability, we know that all pairs of neighbors have many paths from which to choose. What remains is for them to decide in a systematic but local fashion how to choose from among these paths without congesting any edge too much. In the rest of this section, we explore a way to choose paths in this manner.

Take a node v simulated by its neighbor v^b and consider the set $E_{v,b}$ of edges $\{(v^{br}, v^r)\}$. There are $2n^2$ nodes w (all of the form $w = v^{rt}$ or $w = v^{brt}$) which (like v) might potentially use one of the edges in the set as a second edge along a path. Any node which actually does must be simulated by its neighbor across dimension b. The next lemma bounds the number of such nodes.

Lemma 2.5. *For sufficiently large* δ_p *and with high probability, of the* $2n^2$ *nodes at distance* 0 *or* 2 *from either* v *or* v^b, *no more than* $\delta_p n$ *of them are simulated by neighbors across dimension* b.

Proof. As noted before, each node has a probability no more than $\frac{1}{\alpha_p n}$ of borrowing across any given dimension, regardless of the choices made by other nodes. The probability that many nodes choose across the same dimension is no more than

$$\binom{2n^2}{\delta_p n}\left(\frac{1}{\alpha_p n}\right)^{\delta_p n} \leq \left(\frac{2e}{\alpha_p \delta_p}\right)^{\delta_p n}$$

Of course, the actual probabilities depend on the particular $\delta_p n$-size subset we consider and on the relative order in which the nodes of the subset successfully found neighbors to simulate them. Then any node's probabilities are conditioned upon other nodes' previous choices. No matter how these choices are made, however, the stated probabilities are upper bounds on the actual probabilities since when each node chooses it always has at least $\alpha_p n$ choices.

For sufficiently large δ_p, this is smaller than an inverse polynomial in N. ■

Each of the at most $\delta_p n$ nodes (except for v and v^b) can use at most two edges in the set $E_{v,b}$ as a second edge along some path. To use an edge as a second edge, such a node would have to be a neighbor of one of the nodes incident to the edge. If w is of the form $w = v^{rt}$, then w is adjacent to v^r and v^t and no other node incident to an edge in $E_{v,b}$. Similar reasoning applies to nodes w which satisfy $w = v^{brt}$. Trivially, each of v and v^b can use no more than n edges of $E_{v,b}$ as a second edge along some path. If we sum over all edges in $E_{v,b}$ the number of nodes which can use each edge as a second edge counting according to multiplicity, the total will be no more than $(2\delta_p + 2)n$. Therefore no more than $\frac{\eta_p}{4} n$ of these edges will have more than $\gamma_p = \frac{4(2\delta_p + 2)}{\eta_p}$ of those $\delta_p n$ nodes potentially using them as second edges. Let $S'(v, b) = \{r \mid$ more than γ_p nodes can send a path through the edge $(v^{br}, v^r)\}$. Then $|S'(v, b)| \leq \frac{\eta_p}{4} n$.

Let $T(v, v^k, b, b') = S(v, v^k, b, b') - S'(v, b) - S'(v^k, b')$. Then for each adjacent pair of nodes v and v^k, $|T(v, v^k, b, b')| > \frac{\eta_p}{2} n$. The sets $T(v, v^k, b, b')$ will be crucial to our reasoning. The probability that a pair successfully choose a path between them is lower bounded by the probability that they successfully choose the path from $T(v, v^k, b, b')$.

Note that among the edges in all the paths represented by the sets $T(v, v^k, b, b')$, there are now only a logarithmic number of quadruples (w, w^j, c, c') which might potentially congest any given edge. We've already limited the number of paths for which the edge is the second edge along the path. If the edge is the first edge along the path, then one of the edge's endpoints is the simulating node. Each endpoint simulates only a constant number of nodes, and each simulated node contributes exactly n paths. If the edge is the third edge along the path, then the path is simulating an edge at Hamming distance one from the edge considered. There are exactly n edges of this type. The cases in which the edge is the fourth or fifth edge along the path are identical to the first two cases. Thus each edge can be potentially congested by no more than $\mu_p n = (4A_p + 2\gamma_p + 1)n$ paths.

We can now describe Algorithm 2.3, which assigns paths to simulate edges. During Algorithm 2.3, each edge will decide whether or not to accept some path routed through it. Because the other edges in the path simultaneously decide whether or not to accept the path, it is possible that some might accept it while others reject it. If this happens, we assume that an accepting edge counts the path as contributing to its load anyway. Call an edge *saturated* if it has accepted exactly B_p paths routed through it. Otherwise, call it *unsaturated*. Order the pairs (v, v^k) lexicographically. As before, in any round an edge accepts the lowest ordered pairs which try to route through it until it reaches its capacity.

```
for $i = 1$ to $s'_p n$
    for each unassigned adjacent pair of nodes $(v, v^k)$
        $(v, v^k)$ pick a path between them uniformly
        each unsaturated edge agrees to as many
        paths routed through as it can without
        exceeding its capacity, giving preference
        to lower labeled pairs
        all excess pairs remain unassigned
```

Figure 4: Algorithm 2.3

Parallelling what we did before, we will present Algorithm 2.4, a sequential version of Algorithm 2.3. We will show that this modified algorithm terminates having assigned paths between every pair of nodes simulating neighbors, with high probability. Maintaining the parallel with what we proved earlier in this section, we will then show that the two algorithms perform indistinguishably, with high probability. At any time when the pair (v, v^k) attempt to choose a path between them during Algorithm 2.4, let $U(v, v^k, b, b')$ be the subset of $T(v, v^k, b, b')$ consisting of dimensions r for which all of the edges along $P(v, v^k, b, b', r)$ are unsaturated. Define the dedication of a path containing a saturated edge in a fashion similar to the dedication of saturated neighbors before. We dedicate paths to the pair (v, v^k) whenever there are not $\beta_p n$ choices for a simulating path.

```
for $i = 1$ to $s'_p n$
    for all unassigned pairs $(v, v^k)$ in order
        if $|U(v, v^k, b, b')| < \beta_p n$
            dedicate enough $r \in T(v, v^k, b, b')$
        $(v, v^k)$ pick a path between them uniformly
        if the chosen path is unsaturated or dedicated
            $(v, v^k)$ is assigned to the path
        else $(v, v^k)$ remains unassigned
```

Figure 5: Algorithm 2.4

Lemma 2.6. *For a suitably large choice of the constant s'_p, with high probability all pairs of nodes searching for an assignment to a path have been assigned one after $s'_p n$ steps of Algorithm 2.4.*

Proof. Each pair is successfully assigned with probability at least β_p during any step. The rest of the proof is identical to that of lemma 2.1. ∎

We now show that with high probability Algorithm 2.4 never adds dedicated paths with saturated edges to any $U(v, v^k, b, b')$. Thus with high probability Algorithms 2.3 and 2.4 behave identically. This proves that Algorithm 2.3 assigns all necessary paths with high probability.

We will need the following general bound on the sums of random variables from [S].

Lemma 2.7. *Let $\{Y_t\}$ be independent random variables with $Pr[Y_t = 1] = \phi_t$ and $Pr[Y_t = 0] = 1 - \phi_t$. Set $Y = \sum_t Y_t$ and $\phi = \sum_t \phi_t$. Then*

$$Pr[Y > \phi + a] < e^{-a^2/2\phi + a^3/2\phi^3}$$

Lemma 2.8. *With high probability no set $U(v, v^k, b, b')$ ever has cardinality less than $\beta_p n$ at the beginning of some step of Algorithm 2.4, given $\beta_p = \frac{\eta_p}{4}$.*

Proof. There are at most $\mu_p n$ pairs which have a non-zero probability of congesting a given edge on some path represented by an $r \in T(v, v^k, b, b')$. Thus at most $5\mu_p n^2$ pairs have non-zero probability of congesting any of those edges, counting according to multiplicity. For a path to leave $U(v, v^k, b, b')$ one of its edges must become saturated. For $(\frac{\eta_p}{2} - \beta_p)n = \beta_p n$ paths to become unavailable, $B_p \beta_p n$ pairs must choose a path crossing an edge on some path represented by an $r \in T(v, v^k, b, b')$.

The probability that a pair chooses any particular path is at most $\frac{1}{\beta_p n}$, no matter what other choices are made. Thus if there are q_{w,w^j} paths that a particular pair (w, w^j) might choose which contain an edge on some path in $T(v, v^k, b, b')$, then the probability that (w, w^j) chooses such a path is at most $\frac{q_{w,w^j}}{\beta_p n}$, and $\sum_{w,w^j} q_{w,w^j} \leq 5\mu_p n^2$.

Then, by lemma 2.7, the probability that more than $\beta_p n$ paths become unavailable is therefore no more than

$$exp(-\frac{\beta_p}{10\mu_p}(B_p\beta_p - \frac{5\mu_p}{\beta_p})^2 n + \frac{\beta_p^3}{250\mu_p^3}(B_p\beta_p - \frac{5\mu_p}{\beta_p})^3)$$

which, for large enough B_p, is smaller than an inverse polynomial in N. ■

With high probability $O(n)$ steps are sufficient to select all paths. Since we have guaranteed that the paths have constant congestion, this proves the following theorem.

Theorem 2.9. *For each $p < 1 - \sqrt[4]{.5}$ (about .16) there is an $O(\log N)$ step algorithm such that if each of the nodes of an N-node hypercube fails with probability p then with high probability the algorithm finds an embedded fully functioning N-node cube with constant load, dilation and congestion. The paths which simulate the edges of the cube only use live nodes.*

Edge faults are easily handled once node faults are understood. Say each edge fails with probability p_e, each node fails with probability p_n and the failure of any component is independent of the failure of other components. Then the results of this section follow with little change. Specifically, as long as $p_n + p_e - p_n p_e < 1 - \sqrt[5]{.5}$ (about .13), the algorithms of those sections work with high probability. The only addition to our reasoning is that when one node tries to communicate with a neighbor node, it is unsuccessful not only if the neighbor is faulty but also if the link between them has failed.

Another extension of this approach is to the case of arbitrarily large constant probabilities of failure. By showing that most local areas of the cube retain a good structure even when each node fails with probability very close to 1, we can prove that constant delay simulations are always possible.

Theorem 2.10. *Say each node of an N-node hypercube fails independently and with constant probability $p < 1$. Then with high probability, the faulty hypercube can simulate a completely functioning N-node hypercube with only constant slowdown.*

In the preceding discussion, we ignored several details of implementation. For example, we assumed that faulty nodes could participate in the algorithm to search for nodes to simulate them. In reality, other nodes must participate for the faulty nodes. Also, much information must be exchanged by the various participants in the algorithm. It can be shown how to implement this algorithm using polylogarithmic time per step.

3 Fast Routing Around Faults

In this section we examine the problem of routing a permutation on a faulty hypercube. We describe a variant of Valiant-Brebner routing on the hypercube that we call *offset routing*. In order to present our ideas more easily, we first review some basic ideas from Valiant and Brebner's results.

The butterfly is obtained from the hypercube by replacing each node v of the cube by a cycle $(v_0, v_1, \ldots, v_n, v_0)$. We replace each edge (v, v^i) by a pair of edges (v_{i-1}, v_i^i) and (v_{i-1}^i, v_i). We can visualize the set of nodes $\{v_i | v \in H_n\}$ as sharing a *level* of the butterfly. We call edges of the form (v_{i-1}, v_i) *straight* edges and those of the form (v_{i-1}, v_i^i) *cross* edges. All edges connect nodes in adjacent levels or level n to level 0.

We view our hypercube routing as if it took place on the butterfly. A packet starts at some node v_0 and ends at some node w_n. We think of the column of nodes $\{v_i\}$ as shared by the hypercube node v, which assigns each node in the column a different queue from a set of n queues. If a message traverses the straight edge (v_{i-1}, v_i) in some butterfly step, then it is passed from the node v's $(i-1)^{st}$ queue to its i^{th} queue in the hypercube step. If the message traverses the cross edge (v_{i-1}, v_i^i) in some butterfly step, then it is passed from v's $(i-1)^{st}$ queue to v^i's i^{th} queue in the hypercube step. In the remainder of the paper, we will view the routing algorithms as if they take place on the butterfly, although the proofs we give will bound performance on the hypercube as well.

Routing from v_0 to w_n is simplified by the fact that there is a unique path of length n between those two nodes. The i^{th} step in the path connects a node at level $i-1$ with one at level i. If v and w agree in the i^{th} bit, the edge is a straight edge. If they differ, a cross edge is used. For example, to route from the node $(1,1,0)_0$ to the node $(0,1,1)_3$ we would use the path $(1,1,0)_0, (0,1,0)_1, (0,1,0)_2, (0,1,1)_3$.

In the first phase of the Valiant-Brebner routing algorithm, each node in level 0 first sends its packet to a random node in the n^{th} level using the unique path of length n. From there the packet is passed across the straight edge to the level 0 node and then, in the second phase, it is routed along the unique path to its true destination. In [VB] it was shown that this algorithm takes $O(n)$ steps to complete and uses total queue length $O(n)$ at every hypercube node, with high probability.

In the offset routing algorithm, each packet remains fairly close to where the Valiant-Brebner scheme would send it. Its location always differs from where their algorithm would send it by some offset which is a random dimension. Its ability to move easily among the offset nodes will be enhanced by the addition of "jump" edges between the nodes on a given butterfly level. These ideas will be made more concrete in what follows.

A *jump* edge is an edge of the type (v_j, v^i_j). Jump edges are not butterfly e[illegible]s. A packet traversing such an edge would be sent ([illegible] he hypercube) from the j^{th} queue of v across the edge (v, v^i) and deposited in the j^{th} queue of v^i. Note that all n jump edges of the type (v_j, v^i_j) (j varying) are actually manifestations of a single hypercube edge from v to v^i. This means that every hypercube edge is represented $n+2$ times in the butterfly with jump edges: as n different jump edges and 2 cross edges.

Call the path traversed by a packet in the Valiant-Brebner scheme its *virtual* path. In the offset routing algorithm, a packet whose virtual path would go through the node v_{k-1} will pass instead through some node v^i_{k-1}. If its virtual path would leave v_{k-1} via a straight edge, then the offset path will traverse three edges of the type $(v^i_{k-1}, v^{ij}_{k-1}, v^{ij}_k, v^j_k)$. It finds such a path by randomly choosing a dimension $j \neq i$ and attempting to route across the appropriate three edges. If the packet encounters a fault in any of the edges or nodes along those three edges, it returns to the node v^i_{k-1}, which chooses another random dimension and tries again. Note that this means that a packet might have to traverse many more than three edges to pass from one level to the next. If the virtual path would leave v_{k-1} via a cross edge, then the offset path traverses three edges of the type $(v^i_{k-1}, v^{ij}_{k-1}, v^{ijk}_k, v^{jk}_k)$ instead. Note that no matter whether straight edges or cross edges are used in the virtual path, the node ends with a random offset j from its virtual location.

To begin, the message generated by node v repeatedly chooses random dimension j and attempts to route across the edge (v_0, v^j_0) until it successfully finds an initial offset. Say that the message reaches the n^{th} level for the second time with offset i (i.e. it reaches the node w^i_n). Then to conclude, the message finds an offset j for which the path $(w^i_n, w^{ij}_n, w^j_n, w_n)$ is fault-free.

We will prove that the total length of every offset path is $O(n)$ and that every packet traverses its offset path in $O(n)$ steps, with high probability. First we show that few messages ever cross any given small set of edges.

Lemma 3.1. *Take an arbitrary set of h edges on one level of the butterfly. Then with high probability the Valiant-Brebner routing scheme routes only $O(h+n)$ messages through edges in the set.*

Proof. Note that each message can congest at most one edge in the set. The following an alysis applies to the first phase of the routing algorithm. The analysis for the second phase is almost identical.

Say the edges share level l of the butterfly. Then we can partition the butterfly's first l levels into $\frac{N}{2^l}$ nonintersecting butterflies $B_1, B_2, \ldots, B_{\frac{N}{2^l}}$ each built from a subcube with 2^l nodes. For a message to route through one of the h edges, it must start in the same butterfly as the edge. Say that h_i of the edges lie in butterfly B_i. Then each message starting in butterfly B_i has probability $p_i = \frac{h_i}{2^l}$ that it will hit one of the edges. The sum, over all messages, of the probabilities of hitting one o the edges is $\sum_i 2^l \frac{h_i}{2^l} = \sum_i h_i = h$. From lemma 2. the probability that more than $h + \alpha$ messages route through the edges is less than $exp(-\frac{\alpha^2}{2h}(1-\frac{\alpha}{h^2}))$.

If $h > n$, then the probability that more than $(k+1)h$ messages pass through the edges is less than $N^{-\frac{k^2}{4}}$. Similarly, if $h < n$, the chance of having more than $(k+1)n$ messages crossing the set is also less than $N^{-\frac{k^2}{4}}$. ■

Lemma 3.2. *Take an arbitrary set of $O(n^3)$ edges in the butterfly, $i \geq 2$. Then with high probability the Valiant-Brebner routing scheme routes only $O(n^3)$ messages through edges in the set, counting according to multiplicity.*

Proof. Examine each level separately. With high probability if level l has e_l edges from the set, then no more than $O(e_l + n)$ messages will traverse an edge from the set at that level. Summing over all levels, we get that with high probability the number of messages crossing edges from the set is no more than $O(\sum_l e_l + n^2) = O(n^3)$. ■

Consider a hypercube with faulty nodes, the butterfly derived from the cube and a particular node v^i_{k-1} in the butterfly. We will need to know that such a node has ample opportunity to send a packet to the next level. Consider a path $P_{v_{k-1},i,j} = (v^i_{k-1}, v^{ij}_{k-1}, v^{ijk}_k, v^{jk}_k)$. W assume that a message has successfully arrived at v^i_{k-1} and so there are six components - three nodes and three edges - in the path that must all work properly If the probability of failure is less than $1 - \sqrt[6]{\frac{1}{2}}$ and the faults are independent, then each such path has probability less than $\frac{1}{2}$ that it has a faulty component. For subsequent analysis, we would like it to be the case that for all pairs v_{k-1}, i there are at least $\zeta_p n$ dimensions j which lead the message on a functioning path $P_{v_{k-1},i,j}$. We would also like to know that at least $\zeta_p n$ of the paths $P'_{v_{k-1},i,j} = (v^i_{k-1}, v^{ij}_{k-1}, v^{ij}_k, v^j_k)$ are fault-free for all pairs v_{k-1}, i. We also need room to begin and end the routing. That is, we need that $\zeta_p n$ of the paths $Q_{v_0,i} = (v_0, v^i_0)$ and $\zeta_p n$ of the paths $Q'_{v_n,i,j} = (v^i_n, v^{ij}_n, v^j_n, v_n)$ contain no faults. If so many paths are available to all nodes we say the butterfly is *locally routable*.

Lemma 3.3. *If the probability that any component fails is less than $1 - \sqrt[6]{\frac{1}{2}}$ and all failures occur independently, then with high probability the butterfly is locally routable.*

Proof. The set of paths available at any time are node-disjoint. Thus the faultiness of any path is independent of other paths in the set. The proof is therefore similar to that of lemma 2.2. ■

Lemma 3.4. *Say a butterfly has faulty components but is locally routable. With high probability each message in the offset routing traverses a path of length $O(n)$.*

Proof. We will prove that any given message's path is of length $O(n)$ with high probability. Since there are only N messages, this will imply the lemma. Assume that at some point in its route, the packet is at the node v^i, where v is the node it would traverse in the Valiant-Brebner scheme. Assume as well that the packet is scheduled to traverse dimension k. (If the straight edge is to be used or if the packet is at the beginning or end of the route, the analysis is identical.) Then if the packet successfully chooses to jump across dimension j, the path $(v^i_{k-1}, v^{ij}_{k-1}, v^{ijk}_k, v^{jk}_k)$ must have no faults. Since the butterfly is locally routable, $\zeta_p n$ of the possible paths to choose are fault-free. If a faulty path is chosen, no more than six steps are necessary to encounter the fault and to return to v^i_{k-1}. Since a random dimension is chosen at each step, the probability that a packet takes more than $6a(2n+2)$ steps is less than the probability of at least $(a-1)(2n+2)$ heads in a sequence of $a(2n+2)$ tosses of a coin with probability ζ_p of landing tails. This probability is less than

$$\binom{a(2n+2)}{2n+2}(1-\zeta_p)^{(a-1)(2n+2)}$$
$$< \left(\frac{ea(2n+2)}{2n+2}\right)^{2n+2}(1-\zeta_p)^{(a-1)(2n+2)}$$
$$< \left(ea(1-\zeta_p)^{a-1}\right)^{2n+2}$$

an inverse polynomial in N for large enough a. ∎

Now that we know each message moves a distance of $O(n)$ during an offset routing phase, we need to show that its forward movement is delayed by at most $O(n)$ other packets. These facts together will bound the packet's time to its destination. We will show that few other packets choose virtual paths in such a way that they have a non-zero probability of selecting an offset path which congests a given node's path. We will then show that even fewer of those actually congest the path when they use offset paths. Note that a hypercube edge traversed by a given message may be traversed by other messages as either cross edges or jump edges. We consider these two cases separately.

Lemma 3.5. *Consider a set E of $O(n)$ hypercube edges and butterfly straight edges. Let S be the set of butterfly edges such that any packet whose virtual path crosses an edge in S has a non-zero probability of congesting an edge in E as a butterfly edge in its offset path. Then with high probability, there are $O(n^3)$ packets whose virtual paths traverse any of the edges in S, counting a packet several times if it traverses several edges in S.*

Proof. If (w_{l-1}, w^j_l) is a butterfly edge traversed by a packet's offset path then the packet's virtual path must use an edge of the form $(w^{ij}_l, w^{ijl}_{l+1})$ for some pair i, j. There are only n^2 such pairs. The same reasoning would hold if the edge in question were a straight edge. Since $|E| = O(n)$, $|S| = O(n^3)$. By lemma 3.2, only $O(n^3)$ packets traverse edges in S, with high probability. ∎

Lemma 3.6. *Let T be the set of butterfly edges such that any packet whose virtual path crosses an edge in T has a non-zero probability of congesting some edge in E as a jump edge in its offset path. Then with high probability, there are $O(n^3)$ packets whose virtual paths traverse any of the edges in T, again counting according to multiplicity.*

Proof. Say (w_k, w^l_k) is a jump edge traversed by a packet. Let $P_{v_k,i,j}$ or $P'_{v_k,i,j}$ be the path used by the packet when it traverses the jump edge. Then (w, w^l) is either the first or the last edge traversed in the path. If it is the first, then $w_k = v^i_k$, $w^l_k = v^{ij}_k$ and therefore $l = j$. The edge traversed in the virtual path would have been (v_k, v^k_{k+1}) or (v_k, v_{k+1}) for some k. There are n choices for v such that $v_k = w^i_k$ and n choices for k. Thus there are only $O(n^2)$ elements of T whose traversal in some packet's virtual path gives the packet a non-zero probability of traversing the edge (w, w^l) as a jump edge. The same reasoning holds for use of the jump edge as a third edge. Again, since $|E| = O(n)$, $|T| = O(n^3)$. By lemma 3.2, only $O(n^3)$ packets traverse edges in T, with high probability. ∎

Lemmas 3.5 and 3.6 also hold for the set of edges incident to the set of nodes $\{v_k\}$ for some hypercube node v. If we bound the number of packets congesting these edges then we bound the number of packets ever residing in queues in the node v (the queuesize of v). The following two technical lemmas help bound how much congestion actually results from the possible sources.

Lemma 3.7. *Consider a set of nonnegative integers $\{\alpha_{rs} | 1 \le r \le z, 1 \le s \le \sigma(r)\}$ where $\sigma(r) > \zeta_p n$ for all r, $\sum_{rs} \alpha_{rs} \le cn^3$ and $\alpha_{rs} \le n$ for all pairs r, s. If exactly one index s_r is chosen uniformly in $[1, \sigma(r)]$ for each index r then with high probability $\sum_r \alpha_{rs_r} = O(n^2)$.*

Proof. Let $X_r = \alpha_{rs_r}$. We wish to bound the value of $X = \sum_r X_r$. To do so, we bound the moment generating function $M(\lambda) = E(e^{\lambda X})$. We can then bound $Pr[X > bn^2] = Pr[e^{\lambda X} > e^{\lambda bn^2}] \le e^{-\lambda bn^2} E(e^{\lambda X})$. This bound directly follows from Markov's inequality. We will first bound the moment generating functions $M_r(\lambda) = E(e^{\lambda X_r}) = \frac{1}{\sigma(r)} \sum_{s=1}^{\sigma(r)} e^{\lambda \alpha_{rs}}$. We can then use the fact that, since the X_r are independent, $M(\lambda) = \prod M_r(\lambda)$.

If we could find $\alpha_{rx} \le \alpha_{ry}$ and a positive integer δ such that $0 \le \alpha_{rx} - \delta, \alpha_{ry} + \delta \le n$ then by transfering δ units this way we could only increase $M_r(\lambda)$ (for positive λ). This follows because $e^{\lambda \alpha_{rx}} - e^{\lambda \alpha_{rx} - \delta} = e^{\lambda(\alpha_{rx} - \delta)}(e^{\lambda\delta} - 1) < e^{\lambda \alpha_{ry}}(e^{\lambda\delta} - 1) = e^{\lambda(\alpha_{ry} + \delta)} - e^{\lambda \alpha_{ry}}$. By this reasoning, if $A_r = \sum_s \alpha_{rs}$ is fixed, we maximize $M_r(\lambda)$ by setting all terms except possibly one equal to either 0 or n. Thus

$$E(e^{\lambda X_r}) \le \begin{cases} \frac{1}{\sigma(r)}(e^{\lambda A_r} + \sigma(r) - 1) & \text{if } A_r < n \\ \frac{1}{\sigma(r)}(\lceil \frac{A_r}{n} \rceil e^{\lambda n} + \sigma(r) - \lceil \frac{A_r}{n} \rceil) & \text{if } A_r \ge n \end{cases}$$

For the rest of the proof we fix $\lambda = \frac{1}{n}$. If $A_r < n$ then $M_r(\frac{1}{n}) \leq \frac{1}{\sigma(r)}(e^{\frac{A_r}{n}} + \sigma(r) - 1) \leq \frac{1}{\sigma(r)}(1 + \frac{2A_r}{n} + \sigma(r) - 1) \leq 1 + \frac{2A_r}{\zeta_p n^2}$. (The second inequality uses the fact that for $0 \leq \gamma \leq 1$, $e^\gamma \leq 1 + 2\gamma$.) If $A_r \geq n$ then $M_r(\frac{1}{n}) \leq \frac{1}{\sigma(r)}(\frac{2A_r}{n}e + \sigma(r)) \leq 1 + \frac{2eA_r}{\zeta_p n^2}$.

In either case the bound is at most $1 + \frac{2eA_r}{\zeta_p n^2} \leq exp(\frac{2eA_r}{\zeta_p n^2})$. Thus $M(\frac{1}{n}) \leq \prod_r exp(\frac{2eA_r}{\zeta_p n^2}) \leq N^{\frac{2ec}{\zeta_p}}$. Continuing the reasoning of the first paragraph of the proof, $Pr[X > bn^2] \leq e^{-bn} N^{\frac{2ec}{\zeta_p}}$. We can make this probability an arbitrarily large negative power of N by letting b be a large constant. ■

Lemma 3.8. *Consider a set of nonnegative integers $\{\beta_r | 1 \leq r \leq z\}$ where $\sum_r \beta_r = O(n^2)$ and $\beta_r = O(n)$ for all r. Let $\{g_r\}$ be a set of random variables with geometric distributions $g_r \sim G(\zeta_p)$ (i.e. $g_r = a$ with probability $\zeta_p(1-\zeta_p)^{a-1}$) Then with high probability, $\sum_r g_r\beta_r = O(n^2)$.*

Proof. Order the integers by increasing size $\beta_1 \leq \beta_2 \leq \ldots \leq \beta_z$. Then since

$$\beta_{kn+1}, \beta_{kn+2}, \ldots, \beta_{(k+1)n-1}$$

are all at least as large as β_{kn}, we know that $\sum_{k=1}^{\lfloor \frac{z}{n} \rfloor - 1} \beta_{kn} = O(n)$. We assume that $\beta_z = O(n)$, so the sum $\beta_z + \sum_{k=1}^{\lfloor \frac{z}{n} \rfloor} \beta_{kn} = O(n)$.

Now with high probability, all sums $\sum_{r=1}^{n} g_{kn+r}$ are $O(n)$. We know that

$$\sum_r g_r\beta_r \leq \sum_k (\sum_{r=1}^{n} g_{kn+r})\beta_{(k+1)n}$$

Thus, with high probability, $\sum_r g_r\beta_r = O(n^2)$. ■

Theorem 3.9. *If we route using offset routing and the hypercube is locally routable, then with high probability, all packets are delivered in $O(\log N)$ steps and all nodes have total queuesize $O(\log N)$.*

Proof. Focus on the path p_0 of a particular message m_0. We will show that the congestion along p_0 from various sources is $O(n)$ with high probability.

Lemmas 3.5 and 3.6 bound the number of messages which have the potential to congest an edge of m_0's path while passing between levels on their own paths. Enumerate the packets $m_1, m_2, \ldots, m_z$ which have a non-zero probability of congesting p_0 while traversing an edge from an even level to an odd level in their virtual paths. A particular packet may appear several times in the enumeration - once for each even level node along its virtual path from which it might congest an edge of p_0.

The packet m_r has at least $\zeta_p n$ paths which would successfully route it to the next level. Arbitrarily designate exactly $\zeta_p n$ of these paths as special. For the purposes of our analysis, we require m_r to choose a special path before we allow it to route to the next level. This can only increase the amount of congestion placed on any edge, since it increases the number of attempts made by each packet. However, once m_r does choose a special path, we always place it in the last node of the first fault-free path it found. Thus m_r winds up in the same place on the next level as if no special requirements had been made.

Consider the choice of offsets made by the message m_r at even level l_r. Let q_r be the number of choices of pairs of offset dimensions (i, j) for the message m_r which would congest an edge in m_0's path. Then $\sum q_r = O(n^3)$ by lemmas 3.5 and 3.6. ($\sum q_r$ is a second way to count the number of edges in S and T according to multiplicity.)

The choice of the dimension i was actually made for m_r at level $l_r - 1$. The choice was made randomly and uniformly from the set of offsets which led to a fault-free path to level l_r. The exact selection of offsets i are dependent from packet to packet and, for a particular packet, from one level to the next. However, no matter how we condition on previous events, there are always enough offsets to choose from at any given moment. Also, the bounds on the probabilities of congesting p_0 will hold regardless of previous events. Let $i_1 < i_2 < \ldots < i_{\sigma(r)}$, $\sigma(r) > \zeta_p n$, be the choices of offsets at level $l_r - 1$ which lead to a fault-free path to level l_r. Let α_{rs} equal the number of offsets j such that if m_r is routed from level $l_r - 1$ to level l_r using offset i_s and then to level $l_r + 1$ using offset j then congestion results in m_0's path. Then since $\sum_s \alpha_{rs} = q_r$, $\sum_{rs} \alpha_{rs} = O(n^3)$. Since the total number of offsets j is n, clearly $\alpha_{rs} \leq n$. Let i_{s_r} be the offset for m_r actually chosen at level $l_r - 1$. Lemma 3.7 implies that with high probability $\sum_r \alpha_{rs_r} = O(n^2)$.

Set $\beta_r = \alpha_{rs_r}$. At level l_r, whether the message m_r chooses a path from the set of $\zeta_p n$ special paths or the set of $(1-\zeta_p)n$ nonspecial paths, it has at most β_r choices which congest m_0's path. Thus whether we condition whether the choice was special or nonspecial, the probability that message m_r will congest m_0's path is bounded by $\frac{\beta_r}{\zeta_p n}$.

The number of routing attempts made by m_r is $g_r \sim G(\zeta_p)$. On each attempt, the probability that m_r will congest m_0's path is at most $\frac{\beta_r}{\zeta_p n}$. Each attempt is an independent trial and the sum of the probabilities of congestion in the trials is at most $\frac{1}{\zeta_p n}\sum g_r\beta_r$, which is $O(n)$ by lemma 3.8. By lemma 2.7, with high probability $O(n)$ attempts actually did congest m_0's path. Since each attempt involves at most six edges, each attempt can add at most six to the congestion on m_0's path. Thus with high probability, the total congestion on the path from routing attempts at even levels is $O(n)$.

Next examine the congestion on p_0 from other packets beginning and ending their paths. For a packet to congest an edge as the first jump edge of its path, it has to be generated by one of the edge's endpoints. Thus there are at most $O(n)$ such packets. Now consider those packets congesting p_0 during the ending of their paths. Each of the three jump edges used to finish off a path has an endpoint which is at distance one from the virtual destination. Thus at most $O(n)$ packets exist which have the potential to congest any given edge as the first, second or third of these jump edges. Therefore a total of $O(n^2)$ packets have a non-zero probability of congesting some edge of p_0 as they finish their routes. An argument along the lines of the one bounding congestion at even levels shows that congestion from these sources is $O(n)$ as well.

The same argument bounds congestion from routing attempts at odd levels, and also bounds congestion on edges incident to any fixed node. ∎

4 Fault Tolerant Routing

The offset routing algorithm cannot tolerate faults which occur during a particular routing phase. If a packet resides in a node as it fails, that packet is irretrievably lost. Rabin ([R]) discovered how to use the technique of information dispersal to route even in the presence of failing nodes, provided each fault occurs with probability no more than $O(\frac{1}{n^2})$.

In this section we will present a simpler variation of Rabin's algorithm. We also show how our algorithm handles faults occurring with probability $O(\frac{1}{n})$. First, we will briefly sketch the main ideas of the original routing algorithm. Each packet is dispersed into n pieces sent along node-disjoint paths to different locations and then along node-disjoint paths to the final destination.

Since every piece needs to carry $\Omega(n)$ bits of routing information, the original packets must necessarily be large. For concreteness we will assume that all packets contain $m = \Omega(n^2)$ bits. Any piece created will contain $O(\frac{m}{n})$ bits. We also assume that all links and nodes have the capacity to hold a constant number of the original packets (and therefore $\Theta(n)$ pieces).

Rabin proves that with high probability, the number of pieces crossing any node or link never exceeds its capacity. This guarantees that each piece can move during every step and that the entire routing will take no more than $2(n+1)$ steps, $n+1$ steps for each piece to arrive at its random intermediate location and another $n+1$ to arrive at its final destination. No piece's progress is ever delayed by a full queue in the node ahead.

As Rabin points out, routing with dispersal of information can tolerate faults if the dispersal into pieces is done with more redundancy. The pieces may actually be constructed in such a way that the arrival of half (or some other constant fraction) of them is enough to reconstruct the original message. Rabin shows how to do this through matrix multiplication. He then proves that if each link has probability $\frac{1}{n^2}$ of failure, then with probability $1 - 2N(\frac{4e}{n})^{\frac{n}{2}}$ all messages will be safely reconstructed at their destinations.

Our improvement of Rabin's results stems from a more uniform and efficient selection of paths for the routing of pieces. The n pieces are first sent sent to the neighbors of the node which generated the packet. These pieces are then routed along parallel paths to the neighbors of a random intermediate node. From there the pieces are routed along parallel paths to the neighbors of the intended destination, and from there to the destination itself.

If v and w are two hypercube nodes, let $\pi_i(v,w)$ be the path from v^i to w^i used in one phase of the Valiant-Brebner scheme. Let $\Pi(v,w) = \{\pi_i(v,w) | 1 \le i \le n\}$ be the set of all possible such paths. We will first show that if each node v chooses a node v' uniformly and then routes a different piece along each of the n paths in $\Pi(v,v')$ that only $O(n)$ pieces reside in any node's queue at any time step.

Lemma 4.1. *Consider the collection of all paths in the N sets $\Pi(v,v')$ (varying over v), where each hypercube node v has chosen a node v' randomly and uniformly. For any node u and any integer $0 \le j \le n$, with high probability u is the j^{th} node along only $O(n)$ paths in the collection.*

Proof. If u is the j^{th} node along the path $\pi_i(v,w)$ then $u^i = w_1 w_2 \ldots w_j v_{j+1} \ldots v_n$. Separate the two cases in which either $i \le j$ or $i > j$. If $i \le j$, then it must be that $v_{j+1} \ldots v_n = u_{j+1} \ldots u_n$. Precisely 2^j nodes satisfy this condition for v. If one of these nodes chooses a w such that $w_1 \ldots w_{i-1} \bar{w}_i w_{i+1} \ldots w_j = u_1 \ldots u_j$ for some $i \le j$, then u will be the j^{th} node along exactly one path $\pi_i(v,w)$. Otherwise, u will be the j^{th} node along none of the paths $\pi_i(v,w)$, $i \le j$. Thus for each of the 2^j nodes, the probability of exactly one such path is $\frac{j}{2^j}$ and the probability of no such paths is $1 - \frac{j}{2^j}$.

If $i > j$, then $v_{j+1} \ldots v_{i-1} \bar{v}_i v_{i+1} \ldots v_n = u_{j+1} \ldots u_n$ for some $i > j$. Precisely $(n-j)2^j$ nodes satisfy this condition. All reasoning is the same as in the previous case, except now w must be chosen so that $w_1 \ldots w_j = u_1 \ldots u_j$. Thus the probability that u is the j^{th} node along exactly one such path is $\frac{1}{2^j}$. The probability that no path $\pi_i(v,w)$, $i > j$ crosses u in this fashion is $1 - \frac{1}{2^j}$.

We now need only consider the sum of 2^j 0-1 random variables each with probability $\frac{j}{2^j}$ of equalling 1 and $(n-j)2^j$ 0-1 random variables each with probability $\frac{1}{2^j}$ of equalling 1. Call this sum X. Then the moment generating function $M(\lambda)$ for X satisfies

$$M(\lambda) = \left(\frac{j}{2^j}e^\lambda + 1 - \frac{j}{2^j}\right)^{2^j} \left(\frac{1}{2^j}e^\lambda + 1 - \frac{1}{2^j}\right)^{(n-j)2^j}$$

$$= \left(1 + \frac{(e^\lambda - 1)j}{2^j}\right)^{2^j} \left(1 + \frac{(e^\lambda - 1)}{2^j}\right)^{(n-j)2^j}$$

$$\le e^{(e^\lambda - 1)j} e^{(e^\lambda - 1)(n-j)} = e^{n(e^\lambda - 1)}$$

Thus $Pr[X \ge an] \le e^{n(e^\lambda - 1)} e^{-an\lambda} = (e^{e^\lambda - a\lambda - 1})^n$. Setting $\lambda = \ln a$, this implies $Pr[X \ge an] \le$

$(e^{a(1-\ln a)-1})^n$, a bound which can be made as small as desired by increasing the constant a. ∎

The i^{th} piece created from v's packet is sent to v^i, along the path $\pi_i(v,w)$ to w^i and then to w. By lemma 4.1, at no time do more than an pieces cross a given hypercube node, with high probability. Since the packets traversing any link all come from one of the link's endpoints, no more than $2an$ pieces cross the link during any step of the routing. If all links and nodes have the capacity to hold $2a$ original packets, then with high probability no buffering is necessary and no piece waits in a queue.

This analysis assumes that each node routes its packet to a random destination. If we use two phases as in the Valiant-Brebner scheme, the results extend to arbitrary permutation routings.

Theorem 4.2. *If all packets are divided into n pieces which are routed along parallel paths in both phases of the routing algorithm, then for an arbitrary permutation, with high probability the two-phase routing takes $2(n+1)$ steps. No piece waits at any time.*

If we encode the original packet in the pieces via Rabin's matrix multiplication, then we can bound the probability that v's packet is lost by the probability that some $\frac{n}{2}$ of its pieces run into faulty components. But if that many pieces are lost, then at least $\frac{n}{4}$ are lost during one of the two phases of the routing algorithm. Assume they are lost in the first phase; the reasoning for phase 2 is identical. There are at most $(2n+3)n$ different components (nodes or links) encountered by pieces from v during the first phase. We need the following bound on the number of intersections between the routes of different pieces.

Lemma 4.3. *For any hypercube node $u \neq v, w$, no more than two paths in $\Pi(v,w)$ cross u.*

Proof. Count the nodes along the path $\pi_i(v,w)$ starting with v^i as the 0^{th} node. Say that the k^{th} node along $\pi_i(v,w)$ is the same as the l^{th} node along $\pi_j(v,w)$ for $i < j$. Then $w_1^i w_2^i \ldots w_k^i v_{k+1}^i \ldots v_n^i = w_1^j w_2^j \ldots w_l^j v_{l+1}^j \ldots v_n^j$, where $v_q^{q'} = v_q$ iff $q \neq q'$ and similarly for $w_q^{q'}$.

There are four cases. If $k, l \leq j$ then $v_j^j = v_j^i$, a contradiction. Similarly, if $k, l \geq i$ then $w_i^j = w_i^i$, a contradiction. If $k < i, l > j$ or if $l < i, k > j$ then it must be true that $w_i = \bar{v}_i$, $w_j = \bar{v}_j$ and $w_h = v_h$ for $i < h < j$. Thus all $\pi_h(v,w)$ with $i < h < j$ are precluded from crossing u (otherwise $w_h = \bar{v}_h$, a contradiction). Therefore three paths cannot all cross u. ∎

Since no component's failure will affect more than two pieces, it must be true that at least $\frac{n}{8}$ of the $(2n+3)n$ components have failed.

Theorem 4.4. *Given a sufficiently large constant c, if each component of the hypercube fails independently with probability $\frac{1}{cn}$ before or during some permutation routing, then with high probability the routing will be successfully completed. That is, a given packet will arrive at its destination iff both its origin and destination do not fail.*

Proof. Whether or not the i^{th} component fails gives rise to a 0-1 random variable whose moment generating function is $M_i(\lambda) = (\frac{1}{cn}e^\lambda + (1 - \frac{1}{cn}))$. Thus the moment genenating function for the sum of these random variables is

$$M(\lambda) \leq \left(1 + \frac{e^\lambda - 1}{cn}\right)^{(2n+3)n} \leq e^{\frac{(e^\lambda - 1)(2n+3)}{c}}$$

Thus we can bound the probability that more than $\frac{n}{8}$ of the components fail by $exp(\frac{(e^\lambda-1)(2n+3)}{c} - \frac{\lambda n}{8})$. Setting $\lambda = \ln \frac{c}{16}$, we see that the probability of so many failures is no more than $(e^{\frac{1}{4}}(\frac{16}{c})^{\frac{1}{8}})^n$. This bound can be made as low as desired by increasing the constant c. ∎

An increase in the probability of failure by a constant factor can be tolerated by increasing the size of the pieces.

Note that offset routing and information dispersal are complementary techniques. By combining this simplified variant of information dispersal with offset routing, still better results are possible, at least in the theoretic setting. The combined routing algorithm tolerates the failure of a constant fraction of the hypercube's components during the course of the routing of a single permutation. To send a packet, the node first disperses pieces to a well defined set of n nodes at distance 3 (instead of neighbors). The packets are then routed along parallel offset paths to the symmetric set of n nodes close to the destination. Finally, the pieces are combined at the destination. If each node or link fails independently of other components and if in the case it fails it does so at a random time during the routing then this combined algorithm tolerates failure rates of a constant fraction of the hypercube's components.

5 Open Questions

The hypercube is the first network with small node degree which is known to be reconfigurable (with high probability) with only constant slowdown when a constant fraction of its nodes and edges fail. It remains open whether any constant degree network shares this property. In particular, it would be of interest to determine if similar results hold for the butterfly.

6 Acknowledgements

We would like to thank Bill Aiello and Satish Rao for their many helpful comments, and particularly for their help in proving lemma 3.7.

7 References

[A] F. Annexstein, "Fault Tolerance of Hypercube-Derivative Networks," to appear, *Proc. 1st Ann. ACM Symp. on Parallel Algs. and Arch*, June, 1989.

[BS] B. Becker and H.U. Simon, "How Robust is the n-Cube?," *Proc. 27th Ann. IEEE Symp. Foundations Comput. Sci.*, Oct. 1986, pp. 283 - 291.

[DHSS] D. Dolev, J. Halpern, B. Simons, and R. Strong "A New Look at Fault Tolerant Network Routing" *Proc. 16th Ann. ACM Symp. on Theory of Computing,* Apr. 1984, pp. 526-535.

[G] E. Giladi, private communication.

[GHLS] N. Graham, F. Harary, M. Livingston, Q. Stout "Subcube Fault-Tolerance in Hypercubes," unpublished manuscript.

[HLN] J. Hastad, F.T. Leighton and M. Newman, "Reconfiguring a Hypercube in the Presence of Faults," *Proc. 19th Ann. ACM Symp. on the Theory of Computing,* May 1987, pp. 274 - 284.

[R] M. O. Rabin, "Efficient Dispersal of Information For Security, Load Balancing and Fault Tolerance," JACM, to appear.

[S] J. Spencer, **Ten Lectures on the Probabilistic Method**, SIAM, Philadelphia, 1987.

[VB] L. G. Valiant and G. J. Brebner, "Universal Schemes For Parallel Computation," *Proc 13th Ann. ACM Symp. on the Theory of Computing*, May 1981, pp. 263 - 277.